La bibliothèque Xlib

Christian NGUYEN

Departement d'informatique

Universite de Toulon et du Var

Introduction

Si de nombreux *toolkits* existent pour X11, une application graphique comportant une zone de dessin fait souvent appel aux primitives de la couche Xlib.

1. le concept de fenêtres,
2. tracer des lignes et des formes,
3. écrire du texte,
4. couleur et visuels,
5. les évènements.

Les unités

- X Window ne manipule que des unités entières,
- les coordonnées sont définis en nombre de pixels (repère orthonormé, origine supérieure gauche),
- les angles sont définis sur l'intervalle $[0, 360 \times 64]$,
- il n'y a pas de notion d'espace objet ni d'écran normalisé.

Utilisation des fenêtres

L’objet fenêtre

C’est l’objet graphique principal de la Xlib : zone rectangulaire où l’on peut dessiner et récupérer des évènements.

Le serveur X gère chaque fenêtre via un identificateur (entier non signé de 32 bits), contrôle la fenêtre racine *root window* de la hiérarchie de fenêtres, résoud le fenêtrage (*clipping*), contrôle un affichage composé de fenêtres multiples se superposant.

L’objet fenêtre

Pour créer une application fenêtrée simple, les étapes sont :

- connection au serveur,
- contrôle de l’environnement,
- création de la fenêtre,
- affichage de la fenêtre,
- à la fin du programme, destruction de la fenêtre (et fermeture du *display*).

Se relier au serveur

```
Display* affichage;
affichage = XOpenDisplay ((char *) NULL);
```

Le paramètre `NULL` indique l'utilisation de l'affichage indiqué dans la variable d'environnement `DISPLAY`.

Deux possibilités :

1. la connexion a échoué : `NULL` est retournée, le programme devra quitter normalement avec un message d'erreur à l'utilisateur,
2. la connexion a réussie : détermination de lécran auquel on est reliés par la fonction `DefaultScreen`.

```
int ecran;
ecran = DefaultScreen(affichage);
```

Contrôle de l’environnement

Pour être indépendant du matériel, pas d’hypothèses au sujet du type d’écran utilisé.

Demandes au serveur X des caractéristiques de l’écran :

```
DisplayWidth (display, screen)
DisplayHeight (display, screen)
DefaultDepth (display, screen)
```

Créer une fenêtre

```
#include <X11/Xlib.h>

Display* display;
Window parent_window;
int x, y;
unsigned int width, height, border_width;
int depth;
unsigned int class;
Visual* visual;
unsigned long attr_mask;
XSetWindowAttributes* attributes;

window = XCreateWindow(display, parent_window, x, y, width,
  height, border_width, depth, class, attr_mask, attributes);
```

Les paramètres

- `display` obtenu à partir de `XOpenDisplay`,
- `parent_window` : identificateur de la fenêtre parent (la macro-instruction `RootWindow`, donne l'identificateur de la fenêtre racine),
- `x` et `y` donnent l'emplacement du coin supérieur gauche de cette nouvelle fenêtre (une requête plutôt qu'un ordre),
- `width` et `height` définissent la taille de la fenêtre en pixels,
- `border_width` est la largeur du cadre de la fenêtre en pixels,
- `depth` est le nombre de plans (*bit planes*) disponibles (obtenue à partir de la macro-instruction `DefaultDepth`, ou de la constante `CopyFromParent`),
- `class` : constantes `InputOutput`, `InputOnly` ou `CopyFromParent`,
- `visual` décrit un modèle abstrait des capacités du matériel graphique (`CopyFromParent`),
- `attributes` contient 15 zones liées à la fenêtre ; `valuemask` est un masque de bits, indiquant lesquelles des 15 zones sont spécifié réellement.

La structure réelle d'attributs

```
typedef struct {
                    unsigned long  background_pixel;
                    unsigned long  border_pixel;
                    long       event_mask;
                    Bool       override_redirect;
                    ...
                    } XSetWindowAttributes;
```

Nous explicitons seulement quatre de ces attributs :

- `background_pixel` : la couleur du fond de la fenêtre,
- `border_pixel` : la couleur du cadre de fenêtre,
- `event_mask` : la prise en compte de l'événement `expose` est fondamentale,
- `override_redirect` : ignore le gestionnaire de fenêtre (peu convivial).

La structure réelle d’attributs

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>

Display                 *display;
int                     screen;
XSetWindowAttributes    attributes;
unsigned long           attr_mask;

attributes.background_pixel   = WhitePixel(display, screen);
attributes.border_pixel       = BlackPixel(display, screen);
attributes.event_mask         = ExposureMask;
attributes.override_redirect  = False;

attr_mask = CWBackPixel | CWBorderPixel | CWEventMask |
            CWOverrideRedirect;
```

Affichage d’une fenêtre

Il passe par un dialogue avec le gestionnaire de fenêtre via la structure suivante :

```
typedef struct {
                long      flags;
                int       min_width, min_height;
                int       max_width, max_height;
                int       width_inc, height_inc;
                struct {
                          int x; /* Numerator     */
                          int y; /* Denominator   */
                        }  min_aspect, max_aspect;
                int       base_width, base_height;
                int       win_gravity;
                } XSizeHints;
```

Affichage d’une fenêtre

Indicateurs les plus intéressants du masque `flags` :

- USPosition - position spécifiée par l’utilisateur,
- USSize - largeur et hauteur spécifiées par l’utilisateur,
- PPosition - position spécifiée par programme,
- PSize - taille spécifiée par programme,
- PMinSize - taille minimum spécifiée par programme,
- PMaxSize - taille maximum spécifiée par programme,
- PAspect - rapports hauteur/largeur minimum et maximum spécifiée par programme.

Le gestionnaire de fenêtres

Les spécification sont passées au gestionnaire de fenêtres en utilisant la fonction `XSetWMNormalHints` :

```
Display         * display;
Window          window;
XSizeHints      hints;

XSetWMNormalHints(display, window, &hints);
```

Affichage du nom de la fenêtre dans sa barre de titre par le gestionnaire de fenêtre :

```
Display         * display;
Window          window;
char       window_name [ ];

XStoreName (display, window, window_name);
```

Le gestionnaire de fenêtres

Affichage de la fenêtre par l'une des deux fonctions :

```
Display* display;
Window window;

XMapWindow(display, window);
XMapRaised(display,window);
```

Les deux dessineront la fenêtre à l'écran, mais `XMapRaised` demande explicitement à apparaître au-dessus de toutes les autres fenêtres.

X mémorisant tout dans des caches mémoires, on doit vider l'antémémoire par un appel à `XFlush(display);`

Libération des ressources

Une fois notre programme terminé, il reste à détruire toutes les fenêtres et à fermer l'affichage (la connexion au serveur). Les fonctions sont :

```
XDestroyWindow(display, window);
XCloseDisplay(display);
```

Premier exemple de programme

```
#include <X11/Xlib.h>/* X library */
#include <stdio.h>/* std input-output */
#include <X11/Xutil.h>/* X utilities */

int main (int argc, char **argv)
{
Display *myDisplay; /* pointer to the display */
int myScreen; /* screen we are using */
int  myDepth; /* colour plane depth */
XSetWindowAttributes myWindowAttributes; /* window attibutes */
unsigned long myWindowMask; /* mask for window attributes */
Window myWindow; /* window structure */
XSizeHints theSizeHints; /* window hints for window manger */
int                    x  = 200; /* x top left corner of window */
int                    y  = 150; /* y top left corner of window */
unsigned int           width  = 850; /* width of the window */
unsigned int           height  = 700; /* height of the window */
int  border_width = 20; /* border width of the window */
```

Premier exemple de programme

```
myDisplay = XOpenDisplay ("");
if (myDisplay == NULL) {
  fprintf (stderr, "ERROR: No connection to X on display %s\n",
    XDisplayName (NULL));
  exit (0);
}

myScreen = DefaultScreen (myDisplay);

myDepth = DefaultDepth (myDisplay, myScreen);

myWindowAttributes.border_pixel = BlackPixel (myDisplay, myScreen);
myWindowAttributes.background_pixel = WhitePixel (myDisplay, myScreen)
myWindowAttributes.override_redirect = True;

myWindowMask = CWBackPixel | CWBorderPixel | CWOverrideRedirect;
```

Premier exemple de programme

```
myWindow = XCreateWindow(myDisplay, RootWindow (myDisplay, myScreen),
  x, y, width, height, border_width, myDepth, InputOutput,
  CopyFromParent, myWindowMask, &myWindowAttributes);

theSizeHints.flags  = PPosition | PSize; /* set mask for the hints */
theSizeHints.x  = x;      /* x position */
theSizeHints.y  = y;      /* y position */
theSizeHints.width  = width;      /* width of the window */
theSizeHints.height  = height;      /* height of the window */

XSetNormalHints (myDisplay, myWindow, &theSizeHints);
XMapWindow (myDisplay, myWindow);
XFlush (myDisplay);
sleep(10);
XDestroyWindow (myDisplay, myWindow);
XCloseDisplay (myDisplay);
exit (0);
}
```

Dessiner avec X

Toutes les fonctions nécessaires au dessin des formes géométriques de base ont les mêmes paramètres :

- le `display` auquel l'application est connectée,
- un `drawable` qui identifie la fenêtre dans laquelle on dessine,
- un `gc` qui correspond au contexte graphique (*graphic context*),
- les points de contrôle définissant la forme géométrique.

Contextes Graphiques

Ce sont des références à des environnements de dessin : largeur du pinceau, couleur, type de tracé, etc. Une structure de données qui ne doit être manipulée que par l'intermédiaire de fonctions.

Créer un contexte graphique :

```
Display* display;
Drawable drawable; // handle de la fenetre
XGCValues xgcvalues;
unsigned long valuemask;
GC gc;

valuemask = 0L; // valeurs par defaut du CG

gc = XCreateGC (display, drawable, valuemask, &xgcvalues );
```

Valeurs par défaut du CG

arc_mode	ArcPieSlice
background	1
cap_style	CapButt
clip_mask	None
clip_x_origin	0
clip_y_origin	0
fill_style	FillSolid
fill_rule	EvenOddRule
font	implementation dependant
foreground	0
function	GXcopy
graphics_exposures	True
join_style	JoinMiter
line_width	0 (use hardware accelerators, if any)
line_style	LineSolid
plane_mask	all planes (all 1s)
subwindow_mode	ClipByChildren

Modifier le CG

```
Display* display;
int screen;
GC gc;
unsigned long f_colour;
unsigned long b_colour;

f_colour = BlackPixel(display, screen);
b_colour = WhitePixel(display, screen);

XSetForeground (display, gc, f_colour);
XSetBackground (display, gc, b_colour);
```

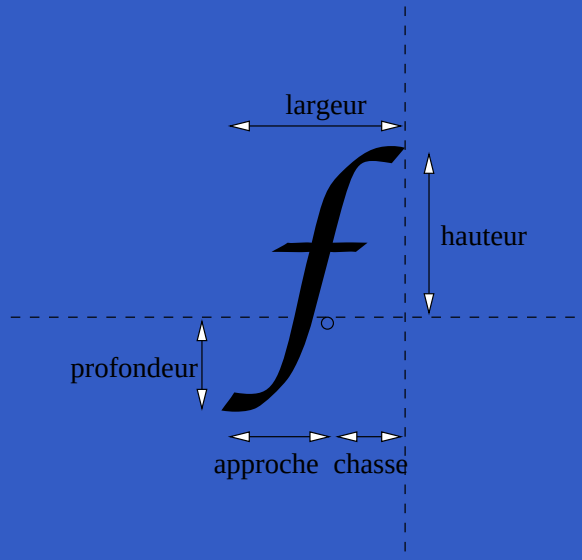
Les formes géométriques de base

```
Display* display;
Drawable drawable;
GC gc;
int x1, y1;
int x2, y2;

XDrawLine (display, drawable, gc, x1, y1, x2, y2);
...
XDrawRectangle (display, drawable, gc, x, y, width, height);
XDrawArc (display, drawable, gc, x, y, width, height,
  start_angle, path_angle);
XFillRectangle (display, drawable, gc, x, y, width, height);
XFillArc (display, drawable, gc, x, y, width, height,
  start_angle, path_angle);
```

Le texte : les polices

Un caractère est définit par 5 paramètres qui sont : hauteur, largeur, profondeur, approche et chasse.



Une police (*font*) définit l'ensemble des caractères ayant la même esthétique.

Le texte : les polices

Répertoire de chaque famille de polices : `/usr/lib/X11/fonts` (`75dpi`, `misc`, ..., et 2 fichiers `fonts.dir` et `fonts.alias`).

Pour assurer la portabilité de X Window, il a fallu décrire de façon très précise les caractères disponibles. Par exemple :

```
-adobe-courier-bold-r-normal-11-80-100-100-m-60-iso8559-1
```

Utilitaires : `xlsfonts`, `xfd`, `xfontsel`.

Charger une police

```
#include <X11/Xlib.h>

Display      display;
XFontStruct  *font_struct;
char         *fontname = times-bold.24;

font_struct = XLoadQueryFont( display, fontname );
if ( font_struct == ( XFontStruct *) NULL)
/* Erreur : la police n'a pu etre chargee.
   On peut essayer d'en charger une autre. */
```

Affichage de texte

Chargement dans un contexte graphique :

```
#include <X11/Xlib.h>
Display         * display;
XFontStruct     * font_struct;
GC          gc;

XSetFont( display, gc, font_struct->fid );
```

Affichage de texte

Affichage et libération de la police :

```
str_size = strlen(string);
XDrawImageString(display, window_id, gc, x, y, string,
  str_size);
XDrawString(display, window_id, gc, x, y, string, str_size);
XFreeFont(display, font_struct);
```

Xlib et la couleur

Parceque X a été conçu pour une large variété d'architecture, les mécanismes de gestion de la couleur sont relativement complexes.

La plupart des écrans couleurs utilisent le modèle RGB pour le codage des couleurs.

Un écran utilise plusieurs bits par pixels (*bit planes*) pour coder la valeur de chaque pixel.

Une *colormap* est utilisée pour transformer la valeur numérique d'un pixel (*pixel value*) vers la couleur effectivement visible à l'écran. C'est en réalité une table située dans la mémoire du serveur dont chaque élément est appelé *colorcell*.

Allocation des couleurs

Un client désirant utiliser une couleur donnée demande l'accès à un colorcell dans la colormap gérée par le serveur X et une pixel value lui est retournée.

Le client : "Quelle colorcell dois-je utiliser pour dessiner en bleu ?"

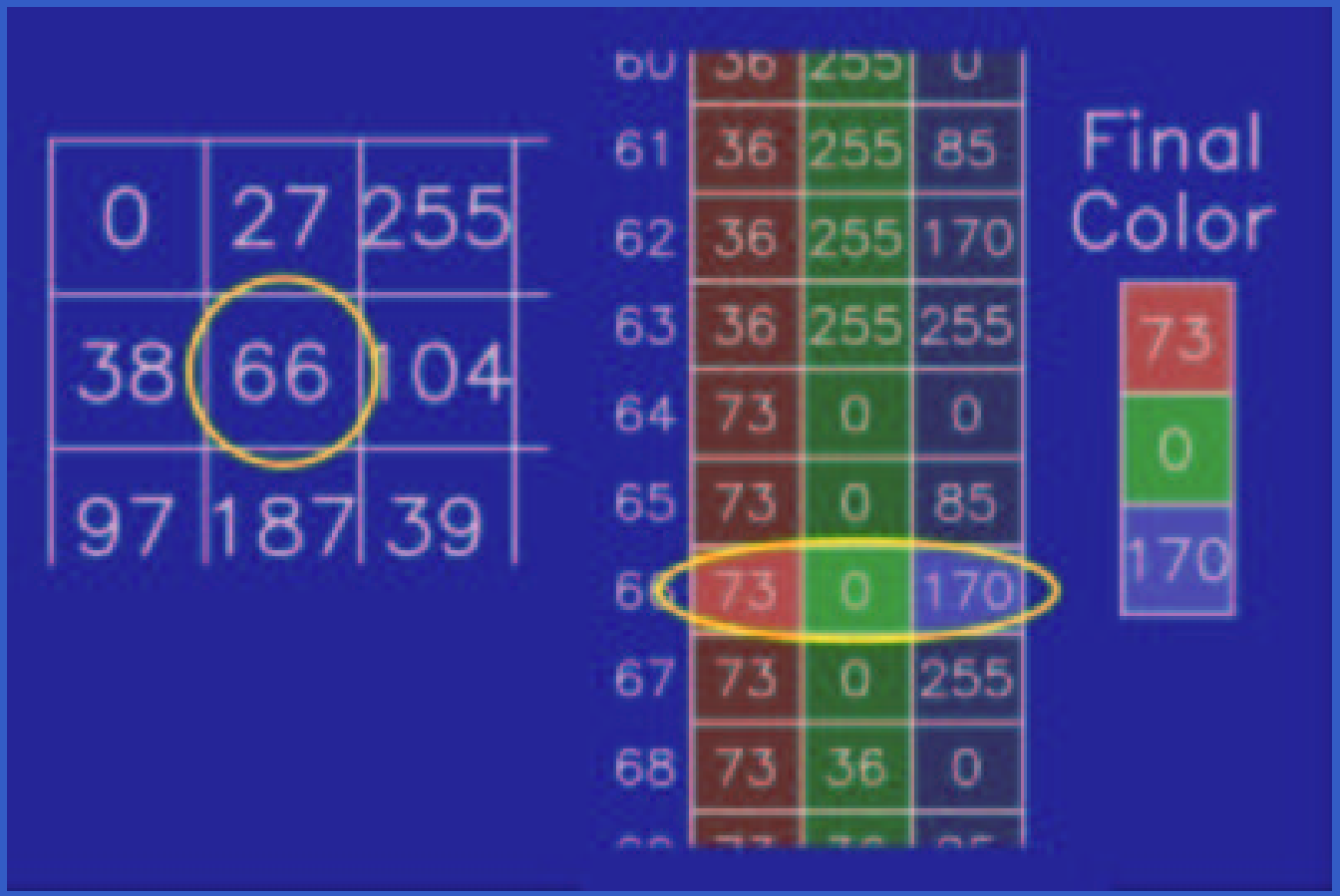
Le serveur : "Tu peux utiliser la colorcell correspondant à cette pixel value"

Couleur et visuels

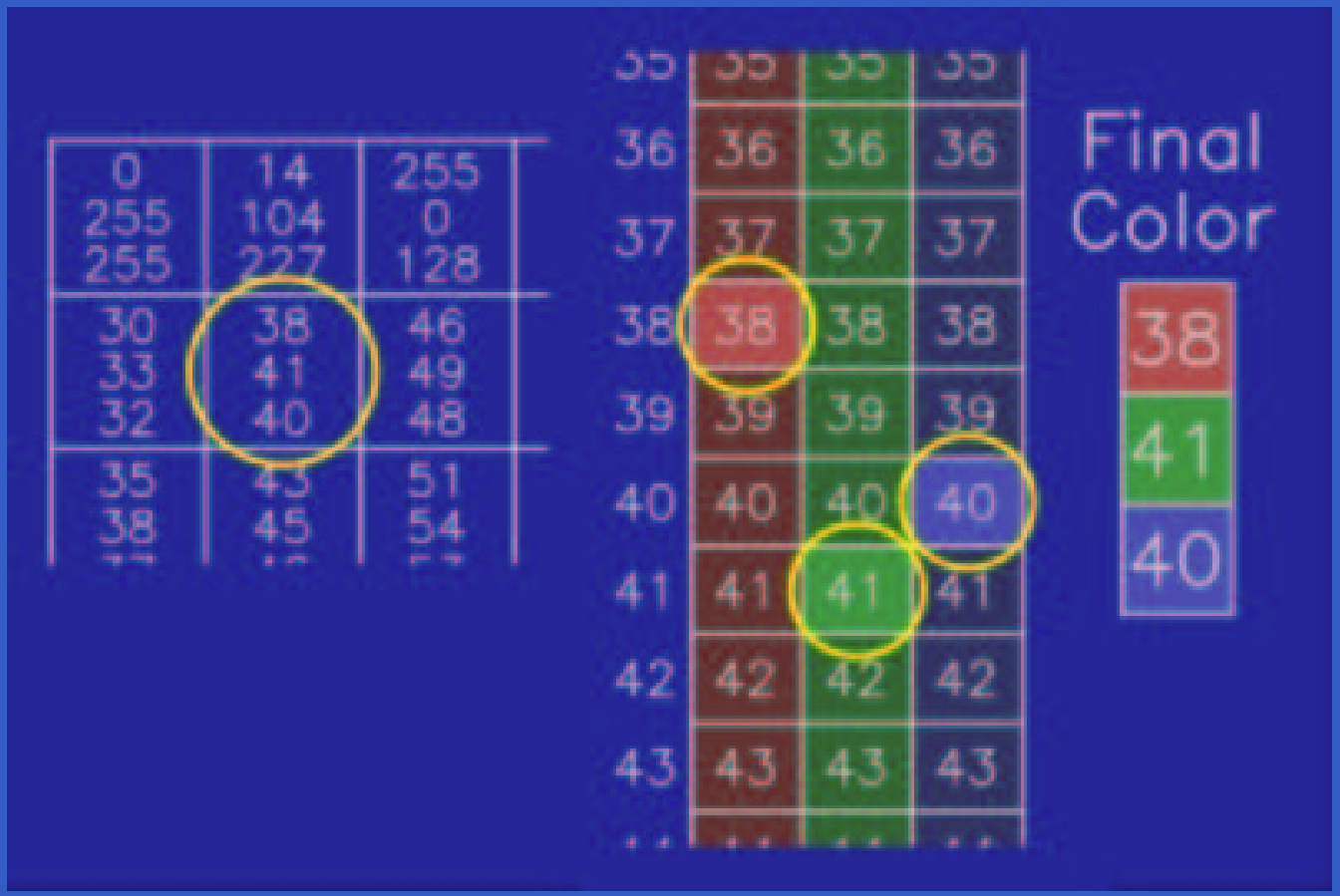
Les “Visuels” sont des abstractions du matériel, il y en a six classes :

- `PseudoColor` : chaque valeur de pixels indexe une table des couleurs RVB,
- `DirectColor` : chaque pixel est décomposé en 3 index associés aux 3 tables de composantes primaires R, V et B,
- `GrayScale` : comme PseudoColor, mais seule l’une des composantes R, V ou B est utilisée,
- `StaticColor` : un système PseudoColor avec une table des couleurs prédéfinie en lecture seule,
- `TrueColor` : un système DirectColor avec une table des couleurs en lecture seule.
- `StaticGray` : un visuel GrayScale avec une table des couleurs en lecture seule.

PseudoColor ou StaticColor



DirectColor ou TrueColor



Couleur et visuels

```
visual = DefaultVisual(display, screen);

if (visual->class == PseudoColor)
  {
    colourmap = DefaultColormap( display, screen );
    depth = DefaultDepth( display, screen );
  }
else
  {
    // visuel ou materiel non PseudoColor
  }
```

Utilisation de la couleur

A partir du nom d'une couleur, la fonction `XLookupColor` permet de récupérer ses composantes RVB depuis la base de données de couleurs `/usr/lib/X11/rgb.txt`.

Cette base de données encourage le partage des couleurs par les applications clientes.

La fonction `XAllocColor` permet ensuite d'introduire ces valeurs dans la table des couleurs (en une seule étape : `XAllocNamedColor`).

Utilisation de la couleur

```
Display   *display;
GC        gc;
Colormap  colourmap;
XColor    req_rgb; /* The RGB specified in rgb.txt */
XColor    hw_rgb;  /* Closest hardware match to the above */
char      *colourname = "blue";
int       status;

status = XLookupColor(display, colourmap, colourname,
                      &req_rgb, &hw_rgb );
if (status == 0) /* Something went wrong */

status = XAllocColor(display, colourmap, &hw_rgb);
if (status == 0) /* Something went wrong */

XSetForeground(display, gc, hw_rgb.pixel);
```

Evénements

Le serveur X transmet uniquement les événements demandés :

- lors de la création d'une fenêtre (`XCreateWindow` prend un paramètre qui est un masque d'événement),
- avec la fonction `XSelectInput` après que la fenêtre ait été créé :

Evénements

```
#include <X11/Xlib.h>

Display         * display;
Window          window;
unsigned long   event_mask;

event_mask = ...;
XSelectInput(display, window, event_mask);
```

Les événements sont pris en compte de façon synchrone ou asynchrone et indiqués par la variable `event_mask`, qui prend la valeur vraie pour les masques exigés. Quelques masques disponibles et leurs événements associés :
`Button1MotionMask/MotionNotify`,
`EnterWindowMask/EnterNotify`, ...

Attente d’événements

Evénements **synchrones** : la fonction `XNextEvent` contrôle la file d’attente d’événement du programme et renvoie le premier événement sinon attend.

Plus restrictifs : `XMaskEvent` et `XWindowEvent` qui renvoient le prochain événement du programme (resp. de la fenêtre) qui apparie le masque d’événement.

Attente d'événements : syntaxe

```
Display         *display;
Window          window;
XEvent          event;
unsigned long   event_mask;

// la fonction la plus utilisee
XNextEvent( display, &event );
switch ( event.type )
  {
    case Expose :          ...
    case ButtonPress :     ...
    case KeyPress :        ...
  }
XMaskEvent( display, event_mask, &event );
XWindowEvent( display, window, event_mask, &event );
XPeekEvent( display, &event );
```

Événements asynchrones

`XFlush` permet de vider la file d'attente des entrées ; `XPending` vérifie s' il y a des événements dans la file d'attente, propres au programme.

`XCheckMaskEvent` et `XCheckWindowEvent` vérifient l'existence d'un événement d'un type particulier. Si un évènement est trouvé, la fonction renvoie la valeur vraie et l'événement sinon elle renvoie faux.

Événements asynchrones

```
Display         *display;
Window          window;
XEvent          event;
int             no_events;
int             type;
int             event_found;
unsigned_long   event_mask;

XFlush( display );

no_events = XPending( display );

event_found = XCheckMaskEvent( display, event_mask, &event);
event_found = XCheckWindowEvent( display, window, event_mask,
                &event);
```

Evénements souris

En fait n'importe quel dispositif de pointage. Des événements "bouton souris" sont produits quand l'utilisateur :

- appuie sur un bouton particulier de la souris,
- libère un bouton de la souris,
- déplace le curseur souris,
- déplace le curseur souris tout en maintenant un de ses boutons dans une fenêtre active ou le descendant d'une fenêtre active.

Evénements souris

Les masques pour prendre en compte les événements souris sont :

Masque	Signification
ButtonMotionMask	déplacement et un bouton pressé
Button1MotionMask	déplacement et le bouton gauche pressé
Button2MotionMask	déplacement et le bouton du milieu pressé
Button3MotionMask	déplacement et le bouton droit pressé
ButtonPressMask	un bouton de la souris a été pressé
ButtonReleaseMask	un bouton précédemment pressé a été libéré
PointerMotionMask	le curseur souris a été déplacée

Evénements souris

Après avoir reçu l'événement on détermine dans quelle fenêtre il s'est produit, ses coordonnées x et y, et l'état des boutons :

```
Window    window;
XEvent    event;
int       x, y;

// evenements bouton
window = event.xbutton.window;
x = event.xbutton.x;
y = event.xbutton.y;

if (event.xbutton.button == 1) printf("Left mouse button\n");
if (event.xbutton.button == 2) printf("Middle mouse button\n");
if (event.xbutton.button == 3) printf("Right mouse button\n");
```

Evénements souris

```
// evenements de mouvement
window = event.xmotion.window;
x = event.xmotion.x;
y = event.xmotion.y;

// evenements MotionNotify : champ bouton non disponible
if (event.xmotion.state & Button1Mask)
   //motion event with left mouse button down
```